CenturaPro™
Visit us at www.ProPublishing.com! Hot Ideas for Centura® Developers

Hosting a Parameter Party

Thomas Althammer

July 1999
Volume 4, Number 7

PRO Publishing

Thomas' article, "De-Sqribe Your Reports," in the May issue covered several aspects of the SQR reporting server. As promised, he now goes hands-on to show you the integration of Centura Team Developer with SQR server. Here he demonstrates a generic technique for constructing parameter entry masks dynamically—pretty nifty for similar products, such as Crystal Reports & Co. If you are still not interested in SQR, go ahead and jump to the paragraph titled "Parameter Party".

As mentioned in my previous article, the SQR server is just an executable. Actually, there are two additional program files, one for printing SPF files (the portable output format of SQR server) and one for executing precompiled reports. These two EXEs represent a subset of the complete functionality—they don't add anything not already in the main program, SQRW.EXE (or SQR.EXE on all non-Windows platforms). One SQR directory for each database brand contains all the necessary components.

To execute a report from the command line, enter the following:

```
SQRW.EXE [report-file]
[database]/[user]/[password] [flags] parameters...
```

Available command line flags include:

- Limiting the number of pages generated, disregard database access.
- Debug information, such as debug level.
- Path and file name of the error file and log file.
- Information on the output, such as a printer-specific file, the portable SQR file format, the Windows printer, HTML, etc.
- For SQLBase, isolation level and other DB-specific information.
- The path for additional include files.

Flags that are needed each time a report gets executed with SQR can be placed in the SQR.INI. One of these standard settings in the SQLBase-specific section of my SQR configuration file is "-LOCKRL", which causes SQR to use the Release Locks isolation level when connecting to the database.

Compilation and runtime parameters in SQR

You can pass arguments to an SQR report by using the ASK or INPUT commands. The value of an INPUT

Continues on page 3

Kingdom COM

Mark Hunter

The release of XSalCOM has caused long-standing issues to catch fire recently. One issue: Can XSalCOM, or CTD itself, scale up to serve many simultaneous users? And a second: Are CTD applications really suited to becoming COM automation servers?

With regard to the first issue: no, XSalCOM won't work well in large-scale distributed transaction situations. The reasons are due to XSalCOM, CTD, and COM itself. For example, XSalCOM produces out-of-process servers, not in-process servers. In other words, when you make a request of an XSalCOM server, an EXE is launched (at least the first time—the EXE might already be running for second and subsequent calls). But an in-process server acts as a DLL, running in the memory space of the caller. So in situations where high throughput is required, out-of-process servers are too slow. XSalCOM works well in low-volume applications, though.

Although the scaling issue is important (indeed, Joe Falcone's column in this issue emphasizes how scalability is driving future product direction), it's out of the immediate control of CTD developers. I really wanted to talk about the second issue, which we can do something about today. There was an active debate in the Centura newsgroups in June about whether most CTD applications are ready to become automation servers. Even if CTD 2.0 existed today, with its improved runtime and other DNA features, would it make sense to COM-enable *existing* CTD applications? Here are some opinions:

From Peter Hazlehurst

"For the last 18 months I have spent all my time in evolving our company's long-term strategy around COM and DCOM and planning how we can evolve our products to be successful in the years to come. Our target is a Web application that can service 10,000 concurrent users, a million sessions a day, and a billion transactions a day.

"I've used SAL to do all manner of things, including writing a Basic interpreter, of all things, so I'm definitely convinced of the basic concept that there are very few things that you can't do in SAL. But there's a big catch, and that is, you have to design whatever you are creating in the right way for it to be successful.

"Why all the controversy with COM? Because COM is not just technology that magically creates a distributed

Continues on page 10

Editor Mark "Magnum PI" Hunter, **Publisher** Dian "Starsky and Hutch" Schaffhauser,
Business Manager Shelley "Columbo" Doyle, **Production Editor** Paul "Cheers" Gould, **Centura Dog** Mocha "Taxi"

Centura Pro (ISSN: 1093-2100) is published monthly (12 times per year) by Pro Publishing, PO Box 2399, Nevada City, CA 95959.

POSTMASTER: Send address changes to *Centura Pro*, PO Box 2399, Nevada City, CA 95959.

Subscription information: To order, call Pro Publishing at 530-265-4082. Cost of domestic subscriptions: 12 issues, $119; Canada: 12 issues, $129. Other countries: 12 issues, $139. Ask about source code disk pricing. Individual issues cost $15. All funds must be in U.S. currency.

Call Centura Software Corp. at 650-596-3400.

If you have questions, ideas for bribing authors, or would just love to chat about what you're doing with Centura products, contact us via one of the means at right.

Contact Us

Centura Pro on the Web
http://www.ProPublishing.com

Editorial Department
Phone: 818-249-1364
Fax: 603-792-2774
E-mail: huntersoftware@netscape.net

Subscription Services
Phone: 530-265-4082
Fax: 530-265-0368
E-mail: shelley@propublishing.com

Mail
Pro Publishing
PO Box 2399
Nevada City, CA 95959

Parameter Party …

Continued from page 1

statement is queried on each report run, whereas ASK is only processed on non-compiled reports. The ASK function can only be placed in the setup section of a report file, beneath the tag, "Declare-Report." The INPUT command can appear anywhere within the report.

Just place the command-line data in the same order as ASK and INPUT commands are executed throughout a report run, one after the other with a space in between. If SQR doesn't find a corresponding value and you haven't specified the "quiet mode," a dialog box pops up and lets the user enter the value. This only works with locally executed reports.

SQR include files are similar to SQLWindows code libraries. They have the same file format as regular SQR programs, and their content is merged with the file they're referenced from during compilation. Include files usually contain commonly used procedures, heading and footing functions, and so on.

So, if you set up different include files, you can use the appropriate one depending on the value entered at the ASK command to compile them for different company letterhead designs, for example. Variable substitution scanning takes place before the #INCLUDE command is processed. This allows you to substitute all or part of the include file name at runtime, adding flexibility to controlling which file is included for the run.

The C interface

In addition to the normal command line interface of the SQR server executable, you can use an external function interface provided in SQRW.DLL. The supported functions are:

```
sqr
sqrcancel
sqrend
```

The command line arguments match the ones mentioned above for the direct call of the server executable.

Although the sqr( ) function works synchronously, you can still cancel the running SQR program with sqrcancel( ) using another thread. (Soon you'll be able to do that with the help of the upcoming Ice Tea Foundation Classes.)

sqrend( ) frees memory and closes cursors left open from previous report executions. The minimum needed for running a report through this interface are five DLLs, a very small footprint, indeed.

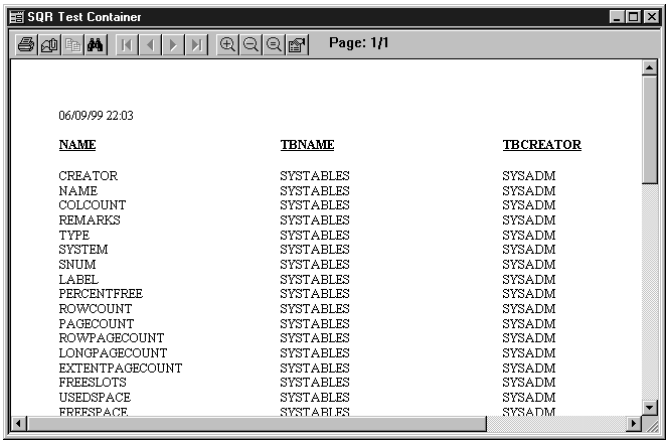

Figure 1. The SQR Viewer ActiveX Control.

ActiveX interface

Sqribe ships a product called "InSQRIBE," a set of three ActiveX components, to integrate with the customer's development environment (for example, Centura Team Developer).

SQR ActiveX Control

In addition to the asynchronous LocalRun( ) function that accepts the same command line flags as the sqr( ) function in the DLL, LocalRunNoWait( ) executes a report asynchronously.

As de-sqribed in my previous article, it's possible to scale up to the third tier with remote execution on an SQR server. The functionality, including TCP/IP and FTP communication handling, is provided by a set of functions in the ActiveX control as shown in **Listing 1**.

Listing 1. Functions exposed by the SQR ActiveX control for remote execution and retrieval of reports.

```
RemoteConnect
RemoteDisconnect
RemoteCancel
RemoteIsConnected
RemotePut
RemoteGet
RemoteDeleteFile
RemoteExec
RemoteExecNoWait
RemoteGetResults
RemoteCheckFile
RemoteGetShellType
RemoteGetServerType
RemoteGetUniqueSqrFileName
RemoteGetUniqueSqtFileName
RemoteSetDebug
```

SQR Viewer ActiveX Control

The SQR viewer is similar to other viewer controls. It lets you display a SPF, optionally with a toolbar.

The functions supported by the control are shown in **Listing 2**.

Listing 2. Functions exposed by the SQR Viewer ActiveX control.

```
Open
Close
SizeTo
ZoomIn
ZoomOut
Copy
Find
PrintSpf
AboutBox
SetCurrentPage
GetCurrentPage
SetCurrentScale
GetCurrentScale
GetPageCount
ShowToolBar
IsToolbarVisible
IsOpen
CanCopy
CanFind
```

As you can see in the listing, there's pretty much everything you need. If you look for a function to save a report, you won't find one. Instead, make a call to the SQR engine via one of the three different techniques described above specifying "–PRINTER:EH" for HTML and CSV and "-PRINTER:PD" for PDF output.

A very nifty feature of the SQR viewer control is the Copy and Find functions. You can search through an SPF file and copy parts of it. Text will be pasted as tab delimited in Excel, for example; graphics and graphs are copied as bitmaps and placed in the Clipboard.

Figure 2 shows the integrated ActiveX viewer within an application.

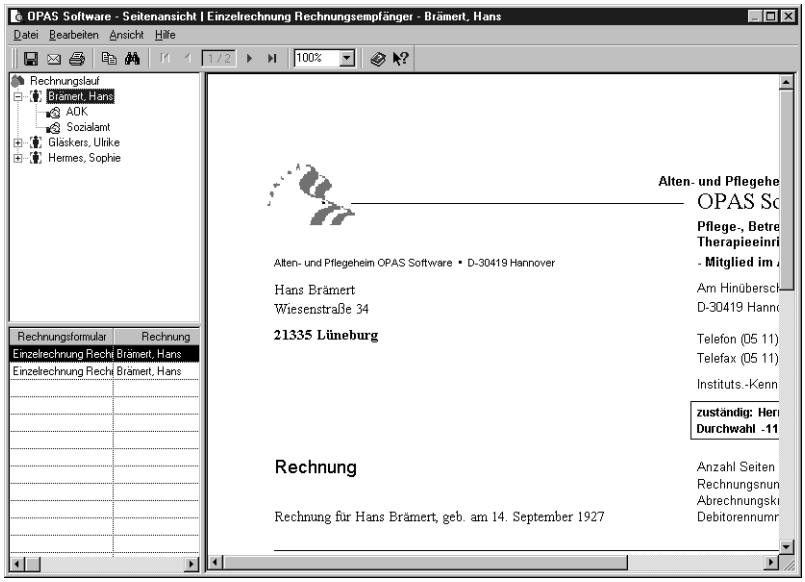

Figure 2. The SQR Viewer ActiveX Control used to display invoices.

SQR Printer ActiveX Control

This is a wrapper around the printing function of the SQR engine. Command line flags and usage work as I've already explained. For printing SPF files displayed in the viewer control, use the PrintSpf( ) function instead.

Parameter Party

Feeding reports with parameters is a pretty easy thing to do as long as you know the necessary data at design time. However, if the user can design and extend reports and use them at different places within the program, certain information might be required to execute a report successfully or decrease the amount of data presented, for example, by passing begin/end dates for a certain range of dates.

As long as there's only one value to be entered, this doesn't really cause any problems—even SQR prompts you for the value in the local set up if not put into "quiet mode." Entering one value after the other in different pop up windows can be quite annoying, though.

Developer and author Gianluca Pivato once presented a technique for creating Windows controls during runtime. You entered the types of parameter requested in his sample application, and the parameter window got generated. Agreed, this is a nice and clean technique. But in my case it's just not suitable, because I have several "business controls," such as tree views and combo boxes with small shortcut fields in front of them. These are controls and groups of controls known to the user from the application, and my parameter window should look similar to the main application window. I was looking for a way to have a dynamically created parameter interface and still use instances of my predefined classes. Dynamic object instantiation of visual controls isn't yet possible with SQLWindows, but there's a work-around . . .

Good old SalCreateWindowEx( )

Does this function ring a bell? Well, it's fairly new (existing since SQLWindows 5.0.2) and got covered several times in earlier issues of *Centura Pro*. Easy trick: set up a class cParameterForm and create one instance for each parameter type/business object you have, as demonstrated in **Listing 3**.

Listing 3. Several possible elements of a parameter mask containing parameter types/business controls.

```
cParameterForm: __pFormText
cParameterForm: __pFormTextLong
cParameterForm: __pFormNumber
cParameterForm: __pFormMoney
cParameterForm: __pFormDate
cParameterForm: __pFormBoolean
cParameterForm: __pFormDrive
```

Each form window is very "short," usually less than two centimeters tall. It contains one or more controls and represents one parameter value. __pFormText, for example, contains a string data field, __pFormDrive a

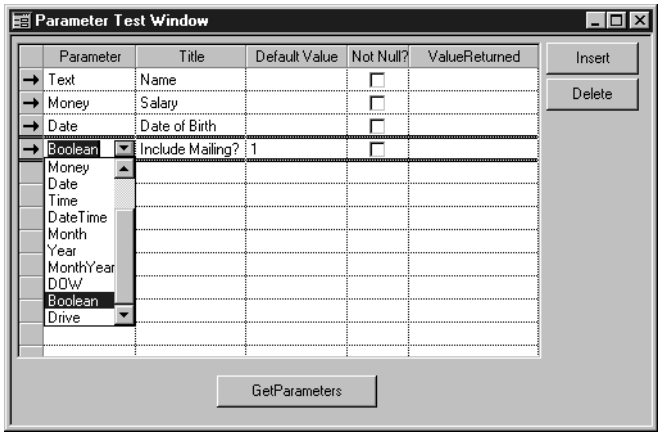

Figure 3. A sample application for defining the dynamically-created parameters. This information could come from the database or a report definition INI file.

combo box for selecting a drive letter. These forms will never be instantiated independently—they're always painted onto a container window.

__dlgParameterSelection is the container and parent of the dynamic parameters created.

Add your own

All you have to do to set up your own custom parameter type is the following:

1. Derive a new form window from the cParameterForm class.

2. Place your control(s) in there, such as table windows, data fields, and pictures.

3. Overwrite the late-bound functions getValue( ) and setValue( ).

4. Add an appropriate PAR_ constant to the user constants section with a unique ID.

5. Set up the corresponding windows name and height in the __dlgParameterSelection.createParameter( ) function—it has to be part of the Select Case statement.

Has the penny dropped? Right, top-level windows can be created at runtime in your empty parameter dialog; you can paint several top-level windows in the client area, one beneath another beneath another. Each contains the controls necessary to handle one of the parameters. Now it's fairly easy to generate your own parameter masks dynamically. However, there are still a couple of obstacles to overcome.

Under the hood

One problem I had to face was the fact that each parameter type isn't necessarily of datatype string. Instead of specifying variants or using different parameter classes, I just generalized the functions getValue( ) and setValue( ) to be of type string. Why? Because in most cases this parameter mask will be used to pass data on the command line or a file to an interface, such as a report. There you need strings anyway, so I didn't bother about all the conversion routines except in the actual cParameterForm instance itself.

The next issue was the tab order. Just create one parameter form after the other as a child of the dialog, I thought. Well, this doesn't work, because the controls would be in reverse order when using the tab key to jump from one to the other. (By the way, all controls in SQLWindows are created in reverse order at runtime.)

So I need to know the height of all previous parameter windows in order to start painting the client area of the dialog from the bottom up, starting with the last parameter. That's why I started to use recursion in the __dlgParameterSelection.createParameter( ) function. **Listing 4** shows the corresponding code.

Listing 4. Recursive call to "createParameter" ensures the correct tab order in the parameter mask.

```
Function: createParameter
  Set bOk = TRUE
  Select Case Parameter[p_nCounter].ParameterType_ID
    Case PAR_Text
      Set sWindow = '__pFormText'
      Set nHeight = 0.3
      Break
    Case ...
  If p_nCounter < p_nParameterCount - 1
    Set bOk = bOk AND createParameter( p_nPosition +
    nHeight, p_nCounter + 1 )
```

Setup parsing

The dialog box used for creating the parameters, __dlgParameterSelection, expects an array of rParameter UDVs as the first parameter and passes the return values back in a receive string array. The structure of rParameter is shown in **Listing 5**.

Listing 5. UDV for the parameter setup passed to __dlgParameterSelection.

```
Functional Class: rParameter
  Description:
  Derived From
  Class Variables
  Instance Variables
    String: Text
    Number: ParameterType_ID
    Boolean: NotNull
    String: DefaultValue
  Functions
```

Depending on the usage and the concept of your application, parameters are retrieved from a text file, from the database, etc. You have to take care of the necessary parsing yourself, by converting a comma-separated list of

Continues on page 12

DNA is in Our Blood:

Topics from the Tropics

Joe Falcone

In my last column I talked about the enormous security problems which we all face as software professionals working with the Internet, and how Centura is making it easy to solve these problems with SQLBase SafeGarde. Likewise, with the software industry focused on developing applications for the Internet, it is my goal to evolve Centura Team Developer into the industry's premier Web 4GL.

After all, when you've got the world's best Windows 4GL, you have a distinct advantage in the race to provide the world's best Web 4GL. Rather than focus on individual near-term point products, we're looking at solutions for the real problems faced by real people in the real world. No one should be surprised that the company that pioneered PC client/server computing will also be the company that leads the way in: securely deploying databases on to the web, redeploying Windows applications on to the Internet, and bringing vertical market solutions to the web.

When we first started working on SQLBase SafeGarde, we encountered those who said, "Security is not a problem"—you just license some encryption algorithms (pay royalties), write some code (pay contractors), and pray that you haven't introduced a boatload of bugs and security holes in the process. But we're in the age of COTS (commercial off-the-shelf) software and reusable components. Why reinvent the wheel when you can buy a proven, tested solution? That solution was the subject of the first Topics from the Tropics column, "Security is Cool, Man".

Likewise when we speak with people about moving their Windows applications on to the web, we hear a variety of stories about what other tools vendors are telling them. The first thing they hear is: "Just rewrite everything in Java, JavaScript, perl and tcl." Well, we know that there are millions of lines of SAL code out there and a mass rewrite is simply impractical for most customers.

Then there are those who suggest moving to an application server platform like Allaire's ColdFusion. But there be demons lurking there as well. This recent *PC Week* article about lack of action to fix security holes in ColdFusion clearly illustrated that Allaire lacks the experience in enterprise software development, deployment and support that Centura brings to the table. In the article Ari Sabah, vice president of technology for NetGrocer Inc. of New York pulled no punches. "Officially, we didn't get anything from [Allaire]," said an annoyed Sabah. "They forgot their customers, and they forgot who got them there."

With customers looking for better alternatives, we could see that the Centura Web Extensions in CWD were the start of something big. We added web capabilities to the world's best 4GL and began the process of evolution that continues in our labs to this day. Today, it is used to create and maintain some pretty good web sites performing a range of activities. Customers have made further extensions to fit into different Internet server environments. And we are integrating technology from the net.db product to produce web business solutions for vertical markets such as e-commerce. Finally, we're updating the run-time architecture to provide increased scalability for Internet environments.

All of this work is ultimately focused on the continued improvement of our Web 4GL capabilities. And that's where DNA comes into the picture.

We joined the Microsoft Distributed interNet Applications Architecture (Windows DNA) partnership program to give Centura access to technology that will be used to further our development of Web 4GL technology. With the addition of Windows NT option packs and the forthcoming release of Windows 2000, Microsoft is providing enterprise quality middleware for transaction processing, message queuing, and other mission critical functions. By aligning our tool and database products with Windows DNA, we will allow all Centura customers to make use of these facilities with the same 4GL ease to which they've become accustomed. And as Internet applications become more sophisticated and move from information delivery to transaction processing, again we will help our customers develop those with our Web 4GL technology and secure SQLBase SafeGarde database.

Continues on page 9

Simple net.db-ing

Sven O. Rimmelspacher

Implementing applications with net.db isn't a tough job at all. In fact, this program may help you eliminate some of those programming chores you've been putting off for no good reason. Here's a practical example.

When people talk about net.db, they often say something like, "Yes, it's a good product, but we and our customers don't have any catalogs or products that we want to show on the Internet, and therefore we have never used it." But when you look at it from another side and think about what net.db can do for you as a system that supports your development issues, you'll find some nice ideas that are easy to implement and can really help you. In this article I'll give you two examples of how we use net.db and how it helps us in the development processes.

How it started

Recently I came to the conclusion that a knowledge base could help in our development cycle. When I thought about how much time I had to accomplish this task, I decided to try a net.db application. (Also, I have never used net.db before and I needed to learn it.) It's simple, can be created in minutes (or at least hours), and can contain everything we need—even business graphics. So I started to create a simple database structure that would fit our needs. It looked like the one in **Listing 1**.

Listing 1. The first table design for the knowledge base.

```
create table knowbase
(
  inputdate datetime not null,    // date/time when entered
  id char(8) not null,            // who has entered it
  category smallint not null,     // in which category fits this entry
  symptom char(254) not null,     // what is the symptom/error/question
  solution char(254)              // what is the solution
);
```

Listing 2. The final table design for the knowledge base.

```
create table knowbase
(
  serial integer not null,        // unique key for each row
  inputdate datetime not null,    // date/time when entered
  id char(8) not null,            // who has entered it
  category smallint not null,     // in which category fits this entry
  symptom char(254) not null,     // what is the symptom/error/question
  solution char(254),             // what is the solution

  primary key (serial)            // definition of primary key
);
create unique index knowbase_1 on knowbase (serial);
```

Some minor problems

I created the table in our SQLBase database in the intranet. Then I started net.db and the page wizard, which led to the error message, "SYSADM.knowbase: This table has no primary key—no page will be created." All right; net.db needs a primary key in order to "wizard" on the table. When I looked at the data structure, I knew I was in trouble because identifying a unique row would lead to a primary key that would be too large (in other words, more than 255 bytes as it is defined in SQLBase). So I had to add an artificial key, which I called serial. For a simple solution to creating a unique key for each new row, I used the sysdbsequence feature of SQLBase. Finally, I had my correct database table design, as shown in **Listing 2**.

More than one table

This time the page wizard was successful and it created my new book.

The ID field should hold the user ID, so we can ask the one who has entered the data if we have further questions. The idea of the category field was to provide the possibility to categorize the data in groups like "error", "question", "feature", "wish", "idea", and so on (these categories should be taken from another table declared below). Finally, the symptom and solution columns were meant to be used for the actual information of the entry.

Searching

The first page I wanted to create was the search page. The default fields were the serial and ID fields. Neither was what I wanted, so I deleted them and created a field for the symptom

column where the user could search. The second search criteria that seemed to make sense was the category; here I wanted to use the category texts (see above) and store the category number, both from the knowcat table. I had no idea how to do this, so I just started the Component Assistant, which led me through this issue. It asked the following:

1. Choose a component from the list? I picked Auto Combo Box.

2. The name, title and data source column? For the column I chose knowbase.category.

3. Which Datasource table should fetch the content of the ComboBox from? I selected sysadm.knowcat.

4. Where to populate the combo box from? It needed to know which value should be written to the data source column and which one should be displayed. So I selected knowcat.category and knowcat.cattext, respectively.

After that I previewed the search page. It contained a combo box with all my values from the knowcat table and an additional [all] entry. Wow! I only had to adjust the position and then I was done!

Browsing

The next step was the browse table. Previewing wasn't possible because no data was found in the table, so I inserted an entry manually. After that the browse table still couldn't be previewed, since net.db immediately shows the detail screen if it finds only one entry for a selection. After inserting a second entry I could finally preview the browse table. It also contained the serial and id columns, which I didn't want; so I removed them and added the category, symptom, and solution columns.

Once again I had the problem that I wanted to display the category texts from the knowcat table; but this time the Component Assistant couldn't create a combo box, since we had just a table. A little browsing through the items in the left pane of the net.db designer showed a Table Joins Wizard. OK. Let's see what this one could do for us . . .

1. I wanted to create a new join. I selected the knowcat table in the Joins to table combo box.

2. The alias isn't important; I just used c.

3. On the next page I had to select which columns make up the join. I selected category from both tables.

Listing 3. The category table for the knowledge base.

```
create table knowcat
(
  category smallint not null,     // the category key
  cattext char(32),               // the text for this category

  primary key (category)          // definition of primary key
);
create unique index knowcat_1 on knowcat (category);

insert into knowcat values ( 1, 'Error' );
insert into knowcat values ( 2, 'Question' );
insert into knowcat values ( 3, 'Wish' );
insert into knowcat values ( 4, 'Idea' );
insert into knowcat values ( 5, 'Feature' );
```

After selecting the properties from the category column and c.cattext (which now existed) in the datasource column, I previewed the browse table again. Now it showed the text instead of the numbers.

Details

My next requirement: to work on the detail screen, which is used for viewing details of a data set and entering new data.

The serial number used for each entry shouldn't be visible, because we can fill it on a new entry automatically. This can be done on the Properties | Advanced page for this field, where you can find a field, "Default text values or SQL sequence." As I've mentioned, we can use the sysdbsequence feature of SQLBase here, so I entered, "SQL:SELECT sysdbsequence.nextval FROM knowbase." This will get the next sequence number from SQLBase each time a new row is to be entered.

Once again I had to create the category combo box, but this time I already knew what to do; it was actually exactly the same as for the search screen.

The last thing was the automatic fill of the date field with the current date when a new data set is created. Once again a SQLBase function helped me here. I just entered, "SQL:SELECT @now FROM knowbase," in the default text field of the properties for this field.

Finally

Now everything was ready to start. I created a link to this page (the URL for this link can be found in the advanced properties of your page), started it, and entered my first data sets. Everything worked fine. I was amazed at how fast and easy this all was. I had worked only for about an hour on this application. I used the rest of the day for designing some header, background, and footer files for these pages and also for a simple description on how to use this stuff, which actually needed much more time than the application itself.

Additional features for this application could include adding a graphical evaluation, say, a GROUP BY category to find out how the entries are categorized or even an image field where you could add screenshots of errors or rapid prototyping ideas.

What next?

After this successful first use of net.db I created other such applications; I think there's one more that could be of interest to you—the SDK constants application.

Often in the newsgroups people ask for the value of an SDK constant. It's sometimes hard to find these values, especially if you don't have a development system, like Microsoft Developer Studio, that comes with them. So I extracted all constants (I found 4,161 of them) from the C header files and put them into a single file in the format:

```
Constant=Value
```

Then I wrote a very simple application that reads this file, splits the lines in a constant and value part, and inserts these tokens in a database table that has the structure:

Listing 4. The table for the SDK constants application.

```
create table sdkconst
(
   name char(64) not null,
   value char(64),
   primary key (name)
);
create unique index sdkconst_1 on sdkconst (name);
```

This time I knew what I had to do. Five minutes later I completed a net.db application that enables the user to browse or select the SDK constants. Since Explorer is usually always running on my desktop, I have immediate access to these values.

Think about it

These applications aren't very sophisticated, but they do show you what you can do with net.db besides the catalog stuff. If you think about what net.db can do for you in your development issues, you'll discover several other applications that really can help you reduce time for jobs that are small and annoying (and therefore not done even if they should be). **CP**

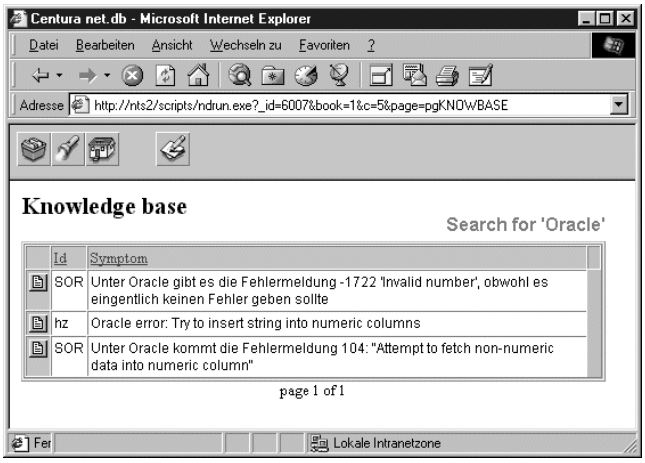

Figure 1. Sample selection in the net.db knowledge base application.

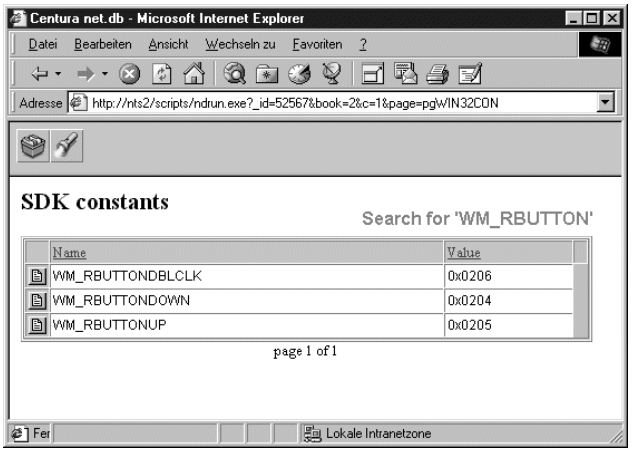

Figure 2. A sample from the SDK application.

Sven O. Rimmelspacher is a senior developer and project manager at Pickert & Partner GmbH, a company that has its own custom quality management system. He is also the author of DocSal, a documentation tool for SAL code. He can be reached at sven@rimpi.de.

DNA is in Our Blood . . .

Continued from page 6

Finally, the survival of a software tool in this day and age depends upon many factors. One essential ingredient is the contribution of quality third-party products. To that end, we're thrilled to finally see the announcement by Pro Publishing and Pivato Consulting of the release of XSalCOM. We view XSalCOM as further confirmation that we're heading in the right direction with our Windows DNA initiative. By extending the COM capabilities of CTD, XSalCOM allows you to take advantage of some Windows DNA technologies today. We would like to encourage other developers to offer CTD extensions to the Centura community, either as licensed products, shareware, or even open source. We'll be devoting a future Topics from the Tropics to the third-party products available for CTD.

This column would not be complete without a tribute to those underdogs on the ice, the Buffalo Sabres. When a seventh seed makes the Stanley Cup finals, it is a small miracle. Here at Centura, we know that we're the underdog as well, but we continue to produce the best 4GL in the industry, another small miracle. But as we all know, it's not whether you win or lose, it's how you play the game.

So please join me in the Centura Community on Thursday, July 22 at 10:00 am Pacific Time (6:00 pm GMT) for a chat about this column. See you then. **CP**

Joe Falcone is Chief Technical Officer of Centura Software Corporation. He can be contacted at joe.falcone@centurasoft.com

Kingdom COM . . .

Continued from page 2

application that magically benefits from reuse and all the rest of the marketechiture hyperbole. COM is hard, COM is about components, and (rightly or wrongly) my suspicion is that much of the SAL code that is out there today will have a hard transition to the COM world. Why? Because, fundamentally, it wasn't designed for a component style use. Most SAL developers, myself included, didn't focus on component style engineering—we focused on solving business problems quickly, and used SAL to do it. SAL lends itself to procedural coding, to forms that have lots of interactions with other forms, to functions that use global variables, and so on. All of these things are a "no-no" in the world of COM.

"Now before I hear that everyone can use certain tools to expose all of SAL's behaviors, you need to realize that this is not the point. The point of COM is small binary compatible objects with no, zero, nil dependencies on other code except through the use of inherited behavior. That is, you write fully self-contained business objects that do their work with no or very limited visual behaviors for server-side applications. The typical SQLWindows SAL application is just not written this way.

"COM is very much this style of application:

```
HRESULT (did it work) =
DoSomething( WithLotsOfInputParameters...,
and MaybeAnOutputParameter)
```

(Note that if you are going to take advantage of COM+ 's Queued Components, you may not use Output Parameters, and must use registered queueable methods that serve as "virtual callbacks.")

"Here are characteristics of most SAL applications that violate the COM model:

- Using forms for data gathering, information processing, and so on.
- MFC-style frameworks of classes that layer in lots of functionality and generate large applications.
- Forms that have local validation, or local SQL statements for validation.
- Global variables between forms and/or pieces of information that need to be "persisted" until you get to the next form.

"For desktop reuse where you are running COM server and client on the same machine, COM style reuse has some value . . . perhaps. But in reality, even if you do have forms and can expose them through other applications written in VB or the like, why would you? What do you get from this? A new UI? Centura's UI is just fine for 98 percent of applications. A new developer environment? Yes; VB is a different world, as is Java, C++ and the like, and if you can call CTD code from there you benefit, I suppose; but do you?

"COM is about components—self-contained ones. SAL applications that have not written functional classes to contain *all* business logic (and I suspect that's most of them) have all the limitations of hauling around a UI that doesn't make sense. Why embed a UI when all you want is the business logic?

"Because SAL makes it easy to write applications that embed logic into the UI, you are in trouble. But it gets worse in SAL than just the UI. COM *hates* having global variables because it makes the object have to stay locked in memory, not shareable for another user. Ever been to a Web site that doesn't allow you to go backwards and forwards through pages? This is the same problem you will have when trying to connect components not designed for "disconnected" use to other applications. COM style objects effectively get connected to the database, inherit behavior from the runtime environment, validate the user's credentials, and so on, for each and every invocation of a method on a newly created component.

"So with all of this, you may be thinking that I believe that doing COM in SAL isn't possible. Well, it is and it isn't. I suspect that we have a lot of work ahead of us to move logic out of UI and into true components. I think that tools like XSalCOM are useful to teach the basic principles and to learn the ideas of the sorts of issues you will be faced with when you componentize your apps. Until Centura fixes the runtime, tools like XSalCOM will be hindered in their ability to execute on their design goals, but if you need it now, go for it.

"What COM has taught me is that you have to think more about your applications. You have to have a conscious design pattern in mind, and execute on it. Above all, don't underestimate the complexities involved with turning a relatively linear, procedural programming based application into a non-linear component-based application. I really do welcome feedback . . . so let's talk." —
Peter Hazlehurst, phazlehurst@phoenixint.com
Vice President, Research & Emerging Technologies
Phoenix International

XSalCOM is not the right technology for Phoenix International, for reasons of scalability. Even CTD 2.0 might not be right. But for those of us whose needs are more modest, Hazlehurst's design arguments are still valid. The cleaner our class designs are, the longer they will last in XSalCOM, CTD 2.0, and beyond.

Let's hear from a few typical developers.

From Martin Knopp

"Regarding the comment, 'Most existing SAL code is not well suited to becoming a COM object, because the code

was not originally designed with COM interfaces in mind,' I do not agree.

"However I think it needs to be discussed in more detail. That comment (plus the related newsgroup postings in June) implies for me that it is a SAL-specific problem that existing code is not well suited to becoming a COM object. I totally disagree with that. However, I do agree that there is a lot of code out there (in SAL and other languages) not suited for becoming a COM object.

"The major problem I see is that most people agree that business logic and presentation layer should be split. But at the point where it comes to coding they 'forget' to do it. For example, we also did that in our earlier projects; these days we have some kinds of business objects now in our SAL code. I guess it would be easy to make COM objects out of these business objects.

"So I would never attempt to use XSalCOM on our earlier projects, as it would simply not make much sense to make some parts of them available via COM. However I could well imagine using XSalCOM to COM-enable parts of our newer projects, for example, to add some Web interfaces.

"I might also use it for COM-enabling existing SAL code to get some business COM objects (out of existing functional classes) which might be used to achieve some kind of three-tier architecture for extensions of existing applications.

"COM-enabling existing visual objects (form windows, MDI windows, table windows) might give you a chance to reuse some existing code even if you did not split business logic from presentation layer, as you could automate the existing windows. I am not too sure about that, as I did not try it myself and therefore do not know exactly how it works. But as far as I can imagine there could be some great potential using that technology to Web-enable existing SAL programs, even if they are not well designed. Think about a form that is used to enter a new customer. I could imagine creating a COM object out of this form, and using this COM object plus an ASP page to implement a Web interface for adding new customers. The same could/should work for querying information using the same technology (create a COM object for the existing form, build an ASP page using this COM object).

"The success of making (true) three-tier apps out of existing SAL code strongly depends on the structure of the existing SAL code. Here I do agree that most code will not qualify without major modifications, as it is not strongly OOP-compliant and will probably not (fully) split business logic from the presentation layer. However, the approach to Web-enabling existing apps by creating COM objects out of the existing visual objects, for example, could be well suited for nearly every SAL project.

"For my own part I see the major potential for XSalCOM in developing new applications as true three-tier applications (maybe having two layers on the client machine but using COM to connect them or by really using an application server and DCOM)."—*Martin Knopp, Pict & Byte, Vienna*

Naturally, I had to ask the guy who started all this:

From Gianluca Pivato

"It all depends on the purpose for the COM object. If it's going to be used on the client side to integrate different applications and environments, you can really use any type of SAL code—the reasons to use COM objects on the client are countless and real; see the entire Microsoft Office suite, VBA, IE, and Selligent's real-world application. You can even use code written entirely using internal functions; it's sufficient to write a functional class that "wraps" around your internals and exposes a higher level functionality. COM classes can share class variables, globals and just about anything else in your CTD app. It's not against the law, and it works perfectly well.

"With XSalCOM the underlying CTD app runs as always, and it can interact with the external world; actually it's the external world that interacts with the CTD app. The same could be valid for server side objects if you start a new instance of the executable for each client.

"You might have design problems when you want a single instance of the executable to serve multiple objects and/or multiple clients. In this case your class design is important, and the use of globals will most likely create a problem unless you make sure that their value is reset to a known state in the XSalCOM constructor or destructor functions.

"In general, since XSalCOM preserves the entire application context, I don't see any major problem related *only* to the style of the code. I see only the advantages of being able to reusing existing code as much as possible.

"The separate problem of high scalability (when needed, of course) is in my view more related to CTD itself. If, according to some, you have to write new code in order to use COM, why use CTD and SAL at all? SAL is not the best example of object orientation, the runtime is slow, and it doesn't support multithreading (2.0 won't either, according to Centura's CTO, in a recent interview in this newsletter). And for non-UI business logic objects SAL is not very productive compared to other mainstream systems.

"My opinion is that XSalCOM is perfectly suited for current CTD capabilities. Almost every XSalCOM limitation is due to CTD itself and/or Centura's decision not to provide source code or documentation in support of my work. In other words, XSalCOM cannot change CTD; it simply makes CTD become a COM automation server. If you need it, it's here now. If you need high scalability components, I suggest you look into more realistic options as C++ or Java and even maybe VB. I always believed that you have to use the right tool for the right job, and CTD is unfortunately but realistically not

the best tool for heavy-duty, distributed components."—*Gianluca Pivato, author of XSalCOM, gianluca@pivato.com, President of Pivato Consulting, Inc., IT Consultant and Founding Member of Ice Tea Group, L.L.C.*

Now it's my turn

I think Peter Hazlehurst was correct in saying that most existing CTD applications are poor examples of COM design. My own applications are a mixture of very good OOP techniques and very old, procedural code. Almost none of them are completely oriented toward components.

So, should my applications become COM automation servers? Yes, absolutely, for the most important of reasons: *If my apps don't start speaking COM soon, management will throw them away* and rewrite everything in VB, or C++, or Net Dynamics, or some other tool. The company needs COM functionality right now. Honestly, the finer points of application design that Hazlehurst discusses above become irrelevant when viewed in this context. I will build automation servers that perform well with perhaps five or 10 simultaneous users, and that will be just fine in many cases. The methods that the servers expose will be of low quality when compared with the COM ideal. But those methods are inside the "black box" of the server, and the COM clients will only care about the results returned, which will be accurate.

I can COM-enable my existing applications for a tiny fraction of the cost of a rewrite. And I can work continuously at revising the code inside the "black box" to produce better COM servers, without needing to change my COM clients at all. CTD 2.0 will also make my servers better, when it arrives.

Some of the existing apps I work on are huge, with hundreds of top-level windows. I will COM-enable a few of these windows, perhaps adding a couple of functions to simulate user mouse clicks, and I will be able to surface the functionality of those windows to Word, or VB, or a Web server. This will increase the value of the app to the rest of the company. This is the single biggest advantage of XSalCOM, in my opinion. At present the integration between my CTD apps and all the other tools in my company is zero. XSalCOM will allow me to quickly increase that integration from zero to some useful number, even if it is not enough to support a high-volume architecture. Others in the company will be aware that they now have a way of reaching the valuable information and logic inside my apps.

As Pivato notes, CTD and XSalCOM are not the right answer for doing completely new, intensive distributed application coding today. Even CTD 2.0 might not be the right answer in such a situation; we'll have to see how much of the Microsoft DNA feature set it supports when it ships. But XSalCOM is a very effective way to COM-enable existing CTD applications that will serve low numbers of simultaneous clients. **CP**

Parameter Party …

Continued from page 5

parameter definitions to the rParameter array. Glance at the sample application and you'll see that it's a pretty easy thing to do.

Hands-on

This article's accompanying source code, ParamParty.APP and ParamParty.APL, isn't a complete library; it's a template to get you going. In the APL I defined a couple of business controls that can be extended and adapted to your class library. The application file contains a form window for selecting the needed parameters and shows the corresponding dialog.

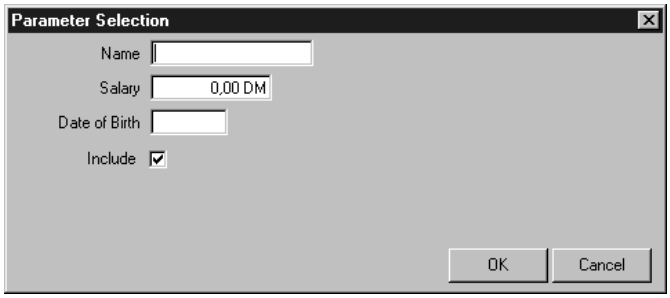

Figure 4. The corresponding parameter mask. The available types of parameters can easily be extended with your own "business controls."

Where to go from here

The technique demonstrated above is pretty simple and straightforward, but still useful. It nicely integrates your application with Crystal Reports, ReportWindows, or SQR reporting server and others. Adapt it for your own work. **CP**

You can download ParamParty.zip from this issue's table of contents at www.ProPublishing.com or find it on this month's Companion Disk.

Based in Hannover, Germany, Thomas Althammer works as an independent IT Consultant and project manager on client/server and Web integration projects. In one of his main projects he develops solutions for social institutions and healthcare associations using Centura products. In addition to being part of the Centura TeamAssist Organisation, he is a founding member of the Ice Tea Group (www.iceteagroup.com), doing consulting for their worldwide clientele. For more information, visit www.althammer.com or send him an email at thomas@althammer.com.